Apple Assembly Line

$1.80

Volume 6 -- Issue 4 January, 1986

New Low Diskette Price

When we first started offering blank diskettes for sale, back in May of 1981, we were able to sell them for the then below retail price of only $50 for 20 diskettes. There have been quite a few changes in this industry in the last five years, and prices have continued to fall. We have a new supplier and can now offer you quality diskettes for only $20 per package of 20, a reduction of 37% since last month.

New Monthly Disks

Since diskette prices have fallen so far, we are now planning to send out disks containing the source code from AAL on a monthly basis, instead of quarterly as we have in the past. See the note on page 11 for the details.

Discover

Sears Financial Network is launching a new credit card this year, called Discover. They offer lower interest rates and no yearly charge to the consumer, and better rates to the merchants as well, so we are pleased to be able to accept this card now. You can use it just like any other card for your phone and mail orders.

Convert Lo-Res Pictures to Hi-res....Bob Sander-Cederlof

Most Apple dot-matrix printer interfaces now include the firmware to print hi-res graphics pictures directly from a screen image. However, most do not provide any way to print lo-res graphics pictures. With the program presented here you can convert a lo-res graphics image into a hi-res picture, ready to be printed by your interface firmware.

Even if you don't ever plan to do such a thing, there are some neat coding tricks in the following program, which you might be able to apply to other hi-res programs.

Lines 1070-1200 demonstrate the use of my lo- to hi-res conversion program. Notice that I started with the label "T". I find I am using that label all the time, lately. I think I started using it as a short mnemonic for "TEST". It is convenient, because in the S-C Macro Assembler environment I can test the program I just assembled by typing "MGO" and the label I want to start at. I find my fingers can now type "MGOT" without my brain even realizing it happened.

The first thing my demo does is call PLOT to create a lo-res picture. I didn't have any real lo-res art around, so I simply drew a 4-by-4 pattern showing all 16 lo-res colors. PLOT fills 16 (4x4) pixels with color 0, 16 with color 1, and so on:

lo-res	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	4	4	4	4	8	8	8	8	C	C	C	C
1	0	0	0	0	4	4	4	4	8	8	8	8	C	C	C	C
2	0	0	0	0	4	4	4	4	8	8	8	8	C	C	C	C
3	0	0	0	0	4	4	4	4	8	8	8	8	C	C	C	C

15	3	3	3	3	7	7	7	7	B	B	B	B	F	F	F	F

The rest of the lo-res screen I did not change, so it will normally show the lo-res equivalent of whatever text was on the screen. Of course if you were really trying to use my CONVERT program you would draw your real lo-res picture.

Lines 1090-1120 turn on the lo-res graphics display, and line 1130 waits until I press a key on the keyboard. After running this much of the program, and studying the dot patterns on the screen, I realized that it is not possible to exactly reproduce the lo-res colors on the hi-res screen (unless I used //e or //c double hi-res). However, by mixing various patterns of dots within the 28 dots (7x4) each lo-res pixel maps to, I can come close to the same color. I don't really know how close I came, because I do not have a color monitor. However, I can at least tell by inspection that all 16 colors map to different dot patterns that will be distinguishable colors.

The PAUSE.FOR.ANY.KEY subroutine will return EQ status if the key I press is RETURN, and NE status if it is any other key. Line 1140 will terminate my test program if RETURN was typed. If it was not RETURN, line 1150 turns on hi-res graphics and line 1160 calls the convert program. Then line 1170 waits for

```
S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners.........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners........................$20
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM.......................................................$30
Source Code for DISASM...................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
"Bag of Tricks", Worth & Lechner, with diskette...........($39.95)  $36 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).......($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL,
 saving you lots of typing and testing.
 The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 20 for $20 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers..........................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)  $50 *
quikLoader EPROM System (SCRG)...............................($179) $170 *
PROmGRAMER (SCRG)..........................................($149.50) $140 *
Switch-a-Slot (SCRG).......................................($179.50) $170 *
Extend-a-Slot (SCRG)..........................................($35)  $32 *

"Programming the 65816", Eyes...............................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)  $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95)  $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Apple ProDOS:  Advanced Features for Programmers", Little..($17.95)  $17 *
"Inside the Apple //c", Little..............................($19.95)  $18 *
"Inside the Apple //e", Little..............................($19.95)  $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Apple ][ Circuit Description", Gayler......................($22.95)  $21 *
"Understanding the Apple II", Sather........................($22.95)  $21 *
"Understanding the Apple //e", Sather.......................($24.95)  $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18 *
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9 *
"Microcomputer Graphics", Myers.............................($12.95)  $12 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16 *
"Assembly Lines -- the Book", Wagner........................($19.95)  $18 *
"AppleVisions", Bishop & Grossberger........................($39.95)  $36 *

          * On these items add $2.00 for the first item and
            $.75 for each additional item for US shipping.
            Foreign customers inquire for postage needed.
      Texas residents please add 6 1/8 % sales tax to all orders.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                 (214) 324-2050                   ***
        *** We accept Master Card, VISA and American Express ***
```

another keypress. Again, RETURN will terminate the test, and any other key will flop back to let me see the lo-res display again. Line 1190 turns the text display back on.

CONVERT is very straightforward. The outer loop, using the X-register, runs from 23 down to 0. This corresponds to the 24 text lines on the screen, or 48 lo-res rows. If your lo-res picture does not use the bottom 4 lines (8 rows), change line 1300 to "LDX #19".

The inner loop, using the Y-register, runs from 39 down to 0, corresponding to the 40 columns of lo-res pixels. Each of the 960 bytes addressed by X and Y contains two lo-res pixels. The top lo-res row (HLIN 0,39 AT 0) is in the low-order nybble of each of 40 bytes starting at $400. The second row (HLIN 0,39 AT 1) is in the high-order nybble of the same 40 bytes. The third and fourth rows are in the 40 bytes starting at $480, and so on. The starting addresses for each row-pair are exactly the same as those for the 24 lines of the text screen. They also happen to be very closely related to the starting addresses for the corresponding rows on the hi-res screen.

I stored the 24 starting addresses in two tables, LOL and LOH. LOL contains the low-half of each address, and LOH the high. Lines 1320-1360 pick up the base address for the current row-pair and put it in pointer LBAS. Lines 1340 and 1370-1380 set up a similar pointer for the hi-res screen. Note that the only difference is that the lo-res screen starts at $400, and the hi-res screen starts at $2000. This address points at the first byte (first seven dots) of the top line of the eight hi-res lines that are in the same position as the lo-res row-pair.

Each lo-res pixel will be mapped to four lines on the hi-res screen, and will be seven dots (or one byte) wide. Each of the 960 lo-res bytes has two pixels, so each byte uses eight lines on the hi-res screen. The right lo-res nybble will be the top four lines, and the left nybble will be the bottom four. After studying the tables of hi-res addresses, I noticed that each set of eight lines follow a very regular pattern. Given the address for the leftmost byte of the top line of a set of eight lines, I can compute the addresses for the next seven lines by successively adding 4 to the high byte of the address. Thus the base addresses for the first eight lines are $2000, $2400, $2800, $2C00, $3000, $3400, $3800, and $3C00. I can always get the base address for the first of the eight by subtracting $400 and adding $2000 to the corresponding lo-res pointer. Line 1370 does that operation in one step with "EOR #$24".

Lines 1400-1480 pick up the current lo-res byte and feed first the right nybble and then the left nybble to PROCESS.NYBBLE. For indexing purposes I multiply the nybble by 8, so that the lo-res color is in the A-register like this: xCCCCxxx. More on that later. Lines 1490-1530 are the south ends of the two nested loops, equivalent to NEXT Y and NEXT X. By the way, please don't get confused by the terms Y and X. They refer in my program to 6502 registers, not Cartesian coordinates. Just to keep your minds nimble, I use the Y-register for the X-coordinate. The X-register is half the lo-res Y-coordinate.

I mentioned above the problem of coming up with patterns of 28 dots to approximate the lo-res colors. There are only six solid hi-res colors, which correspond exactly to lo-res colors 0, 3, 6, 9, 12, and 15. The other 10 lo-res colors take double the normal hi-res resolution to reproduce exactly. However, as Don Lancaster explains in detail in his "Enhancing Your Apple II -- Volume I", you can produce thousands of shades in hi-res by using dot patterns. I picked 12 of his patterns based on the names he gave them, since I did not have a color monitor. His patterns fit in a 28-dot by two line array. Since each byte stores seven dots, it takes 28 dots before the some of the patterns repeat. Using two lines with different or offset patterns gives even more variety.

The table SHADES in lines 1900-2050 give sixteen patterns. The first four bytes of each color are for the first line of 28 dots, and the other four bytes give the second line of 28 dots. Each lo-res pixel will use only one pair of bytes from the set of eight, depending on which column it is in. The last two bits of the lo-res column number (in the Y-register) select which byte pair to use.

Lines 1650-1700 build an index to the byte pair by merging those two bits with the color*8. Then by addressing "SHADE,X" I get the first byte of a pair, and by using "SHADE+4,X" I get the second one. Each lo-res pixel will use the four hi-res bytes by repeating the pair selected from SHADES.

The rest of the code in PROCESS.NYBBLE involves putting the selected color bytes into the hi-res area. HBAS points at the top line of the four to be stored into, and the Y-register points to the byte on that line; so "STA (HBAS),Y" will store into that byte. COMMON.CODE (so named because of a lack of creativity on my part this morning when I discovered that the same eight lines appeared twice) gets and puts two color bytes. The first byte goes into (HBAS),Y; then I add 4 to the high byte of HBAS (since I KNOW it is zero, ORA can be used to add the bit) and store the second byte at the new (HBAS),Y. The "EOR #$0C" at line 1720 changes $24 to $28 or $34 to $38. Similarly, the "EOR #$1C" at line 1750 changes $2C to $30 or $3C to $20. This last possibility leaves HBAS prepared for the next column, automatically!

Some of the same tricks could be used in writing a program to copy text from the text screen to the hi-res graphics screen, or for a general purpose routine to write characters onto the hi-res screen. Instead of using a color map, we would need a table of dot-matrix characters. Maybe this is just how everyone does it, but I don't recall seeing all of these tricks in any previous code. Especially the idea of getting the hi-res base pointer by merely toggling two bits in the equivalent text pointer, and the idea of generating successive hi-res pointers by merely adding 4 to that base pointer.

When I wrote this program I wasn't really worrying about speed or space. Nevertheless, as you can see, it is fairly compact. As for speed, it takes less than a second.

```
                  1000 *SAVE S.LORES TO HIRES
                  1010 *--------------------------------
26-               1020 LBAS    .EQ $26,27
2A-               1030 HBAS    .EQ $2A,2B
2E-               1040 SAVEX   .EQ $2E
30-               1050 COLOR   .EQ $30
                  1060 *--------------------------------
                  1070 T
0800- 20 37 09    1080         JSR PLOT
0803- AD 50 C0    1090         LDA $C050    GRAPHICS
0806- AD 52 C0    1100         LDA $C052    SOLID (40 BY 48 PIXELS)
0809- AD 54 C0    1110         LDA $C054    PRIMARY PAGE
080C- AD 56 C0    1120 .1      LDA $C056    LO-RES
080F- 20 23 08    1130         JSR PAUSE.FOR.ANY.KEY
0812- F0 0B       1140         BEQ .2       ...<RETURN>
0814- AD 57 C0    1150         LDA $C057    HIRES
0817- 20 2E 08    1160         JSR CONVERT
081A- 20 23 08    1170         JSR PAUSE.FOR.ANY.KEY
081D- D0 ED       1180         BNE .1       ...NOT <RETURN>
081F- AD 51 C0    1190 .2      LDA $C051    TEXT
0822- 60          1200         RTS
                  1210 *--------------------------------
                  1220 PAUSE.FOR.ANY.KEY
0823- AD 00 C0    1230 .1      LDA $C000    WAIT FOR ANY KEY
0826- 10 FB       1240         BPL .1       ...NOT YET
0828- 8D 10 C0    1250         STA $C010    CLEAR STROBE
082B- C9 8D       1260         CMP #$8D     SET .EQ. IF <RETURN>
082D- 60          1270         RTS
                  1280 *--------------------------------
                  1290 CONVERT
082E- A2 17       1300         LDX #23      OR #19 IF MIXED MODE
0830- A0 27       1310 .1      LDY #39      COLUMNS 0...39
0832- BD 73 08    1320         LDA LOL,X    SET UP BASE POINTER FOR LINE
0835- 85 26       1330         STA LBAS
0837- 85 2A       1340         STA HBAS     SAME FOR HI-RES
0839- BD 5B 08    1350         LDA LOH,X
083C- 85 27       1360         STA LBAS+1
083E- 49 24       1370         EOR #$24     SHIFT FROM $400 TO $2000 FOR HI-RES
0840- 85 2B       1380         STA HBAS+1
0842- 86 2E       1390         STX SAVEX    SAVE X-REG
0844- B1 26       1400 .2      LDA (LBAS),Y GET TWO LO-RES PIXELS
0846- 48          1410         PHA          SAVE FOR LOWER ONE
0847- 0A          1420         ASL          UPPER PIXEL * 8
0848- 0A          1430         ASL
0849- 0A          1440         ASL
084A- 20 8B 08    1450         JSR PROCESS.NYBBLE
084D- 68          1460         PLA          GET LOWER PIXEL
084E- 4A          1470         LSR          TIMES 8
084F- 20 8B 08    1480         JSR PROCESS.NYBBLE
0852- 88          1490         DEY          NEXT COLUMN. SCANNING RIGHT TO LEFT
0853- 10 EF       1500         BPL .2       ...ANOTHER ONE
0855- A6 2E       1510         LDX SAVEX    RESTORE X-REG
0857- CA          1520         DEX          NEXT LINE, SCANNING BOTTOM TO TOP
0858- 10 D6       1530         BPL .1       ...ANOTHER ONE
085A- 60          1540         RTS          FINISHED!
                  1550 *--------------------------------
085B- 04 04 05
085E- 05 06 06
0861- 07 07       1560 LOH     .HS 04.04.05.05.06.06.07.07  HIGH BYTES
0863- 04 04 05
0866- 05 06 06
0869- 07 07       1570         .HS 04.04.05.05.06.06.07.07  OF SCRN PNTRS
086B- 04 04 05
086E- 05 06 06
0871- 07 07       1580         .HS 04.04.05.05.06.06.07.07  (TEXT OR LO-RES)
                  1590 *--------------------------------
0873- 00 80 00
0876- 80 00 80
0879- 00 80       1600 LOL     .HS 00.80.00.80.00.80.00.80  LOW BYTES
087B- 28 A8 28
087E- A8 28 A8
0881- 28 A8       1610         .HS 28.A8.28.A8.28.A8.28.A8  OF SCRN PNTRS
0883- 50 D0 50
0886- D0 50 D0
0889- 50 D0       1620         .HS 50.D0.50.D0.50.D0.50.D0
```

```
                   *--------------------------------
                   PROCESS.NYBBLE
088B- 29 78               AND #$78      MASK THE SHIFTED NYBBLE
088D- 85 30               STA COLOR
088F- 98                  TYA           LO-RES COLUMN
0890- 29 03               AND #3        LOW 2 BITS
0892- 05 30               ORA COLOR          0CCCC0YY
0894- AA                  TAX
0895- 20 A4 08            JSR COMMON.CODE
0898- 49 0C               EOR #$0C           3RD LINE OF 4
089A- 85 2B               STA HBAS+1
089C- 20 A4 08            JSR COMMON.CODE
089F- 49 1C               EOR #$1C           NEXT LINE
08A1- 85 2B               STA HBAS+1
08A3- 60                  RTS
                   *--------------------------------
                   COMMON.CODE
08A4- BD B7 08            LDA SHADES.X       EVEN LINE
08A7- 91 2A               STA (HBAS),Y
08A9- A5 2B               LDA HBAS+1
08AB- 09 04               ORA #4
08AD- 85 2B               STA HBAS+1
08AF- BD BB 08            LDA SHADES+4,X     ODD LINE
08B2- 91 2A               STA (HBAS),Y
08B4- A5 2B               LDA HBAS+1
08B6- 60                  RTS
                   *--------------------------------
08B7- 00 00 00
08BA- 00 00 00
08BD- 00 00        SHADES .HS 00.00.00.00.00.00.00.00  0--BLACK
08BF- AA D5 AA
08C2- D5 55 2A
08C5- 55 2A               .HS AA.D5.AA.D5.55.2A.55.2A  1--MAGENTA
08C7- 91 A2 C4
08CA- 88 C4 88
08CD- 91 A2               .HS 91.A2.C4.88.C4.88.91.A2  2--DARK BLUE
08CF- 11 22 44
08D2- 08 44 08
08D5- 11 22               .HS 11.22.44.08.44.08.11.22  3--PURPLE
08D7- 2A 55 2A
08DA- 55 2A 55
08DD- 2A 55               .HS 2A.55.2A.55.2A.55.2A.55  4--DARK GREEN
08DF- 33 66 4C
08E2- 19 4C 19
08E5- 33 66               .HS 33.66.4C.19.4C.19.33.66  5--GRAY 1
08E7- D5 AA D5
08EA- AA D5 AA
08ED- D5 AA               .HS D5.AA.D5.AA.D5.AA.D5.AA  6--MEDIUM BLUE
08EF- DD BB F7
08F2- EE F7 EE
08F5- DD BB               .HS DD.BB.F7.EE.F7.EE.DD.BB  7--LIGHT BLUE
08F7- A2 C4 88
08FA- 91 88 91
08FD- A2 C4               .HS A2.C4.88.91.88.91.A2.C4  8--BROWN
08FF- AA D5 AA
0902- D5 AA D5
0905- AA D5               .HS AA.D5.AA.D5.AA.D5.AA.D5  9--ORANGE
0907- B3 E6 CC
090A- 99 CC 99
090D- B3 E6               .HS B3.E6.CC.99.CC.99.B3.E6  A--GRAY 2
090F- D5 AA D5
0912- AA AA D5
0915- AA D5               .HS D5.AA.D5.AA.AA.D5.AA.D5  B--PINK
0917- 6E 5D 3B
091A- 77 3B 77
091D- 6E 5D               .HS 6E.5D.3B.77.3B.77.6E.5D  C--LIGHT GREEN
091F- 2A 55 2A
0922- 55 AA D5
0925- AA D5               .HS 2A.55.2A.55.AA.D5.AA.D5  D--YELLOW
0927- 2A 55 2A
092A- 55 D5 AA
092D- D5 AA               .HS 2A.55.2A.55.D5.AA.D5.AA  E--AQUAMARINE
092F- 7F 7F 7F
0932- 7F 7F 7F
0935- 7F 7F               .HS 7F.7F.7F.7F.7F.7F.7F.7F  F--WHITE
```

```
                 2060 *-------------------------------
                 2070 *   FILL CORNER WITH SAMPLES OF EACH COLOR
                 2080 *-------------------------------
0937- A0 00      2090 PLOT    LDY #0
0939- 84 30      2100         STY COLOR
093B- A2 03      2110 .1      LDX #3
093D- A5 30      2120 .2      LDA COLOR      00, 44, 88, CC
093F- 99 00 04   2130         STA $400,Y     GR ROWS 0-3
0942- 99 80 04   2140         STA $480,Y
0945- 18         2150         CLC
0946- 69 11      2160         ADC #$11       11, 55, 99, DD
0948- 99 00 05   2170         STA $500,Y     GR ROWS 4-7
094B- 99 80 05   2180         STA $580,Y
094E- 69 11      2190         ADC #$11       22, 66, AA, EE
0950- 99 00 06   2200         STA $600,Y     GR ROWS 8-11
0953- 99 80 06   2210         STA $680,Y
0956- 69 11      2220         ADC #$11       33, 77, BB, FF
0958- 99 00 07   2230         STA $700,Y     GR ROWS 12-15
095B- 99 80 07   2240         STA $780,Y
095E- C8         2250         INY
095F- CA         2260         DEX
0960- 10 DB      2270         BPL .2
0962- 69 11      2280         ADC #$11       .., 44, 88, CC, END
0964- 85 30      2290         STA COLOR
0966- 90 D3      2300         BCC .1         ...MORE
0968- 60         2310         RTS
                 2320 *-------------------------------
```

A Question About BRUN

Mike Lawrie, a reader in South Africa, reports that he tried our prime benchmark (Sep 85 AAL) in a Titan Accelerator card equipped with a 65802. It ran 1000 times in 41 seconds, which correlates very nicely with my predictions in the article. The Titan card runs at 3.58 MHz, and I predicted .35 seconds for 10 repetitions at 4 MHz.

Mike also asked an interesting question, which has been asked by a lot of you at one time or another. Why is it that some assembly language programs can be BLOADed and CALLed, but not BRUN? Even the following very simple program will not return from a BRUN, while it will from a BLOAD followed by a CALL:

```
       JSR $FF3A   Ring the bell
       RTS
```

The problem is inside the DOS BRUN command. This command does not use a JSR command to jump to the binary code just loaded. Rather, it uses a JMP command. No return address is left on the stack. When the RTS at the end of the program is executed, it pops garbage off the stack and returns wherever that garbage indicates. What will happen is rather unpredictable.

The Applesoft CALL command does use JSR, and so it works. So does the monitor G command, and the S-C Macro Assembler MGO command. In ProDOS, the BRUN processor works correctly, using a JSR.

This leaves the question: How should a BRUNnable program end under DOS 3.3? If it is to return to the command prompt (] for Applesoft) then the last line should be JMP $3D0. If the BRUN command came from a machine language program (unlikely) then the called program should end with a JMP to a known entry point in the calling program. The most likely case is an Applesoft program that uses a machine language routine. The best way to handle this is to use BLOAD and CALL.

Monthly AAL Source Disk Subscriptions Now Available

We have always made the source code of all the programs published in Apple Assembly Line available on disks. We have collected the programs from three issues together on Quarterly Disks, priced at $15 each or $45/year.

Now that diskettes are so much less expensive, we have decided to try another approach. For those who are interested in getting the source code on disk, we would like to send the source disk along with each newsletter. We will still collect three issues onto Quarterly Disks, for late comers. But those of you who have Quarterly Disk subscriptions will start getting the Monthly Disks. We will send the disk and newsletter in the same envelope, First Class Mail.

The price for combined newsletter/disk subscriptions will be $64 in the USA, Canada, and Mexico. For other countries the postage is higher, so the fee will be $87.

If you want to synchronize your newsletter and Monthly Disk subscriptions, you can pro-rate the Monthly Disk at $3.75 per month ($4.75 for overseas). You can check the length remaining on your current newsletter subscription by looking at the mailing label: the number in the upper-right corner is the year and month of the last issue of your current subscription.

Text File Transfer Using DOS 3.3 File Manager.....Bob Potts

Transferring text files from one drive to another can be frustrating and time consuming. The standard procedure is to read from the file on the source drive and write to the file on the target drive. One possible solution is to use FID, but you must BRUN FID and cannot use it from inside an Applesoft program.

With this in mind, I set out to write a utility which will transfer a text file using the DOS 3.3 File Manager (FM) routines. FM has been a part of every release of DOS, but very little documentation has been written about these powerfull routines. While RWTS concerns itself with tracks and sectors, FM deals with whole files, be they binary, text, or Applesoft. I recalled that a couple of years ago, Bob Sander-Cederlof had assisted me with a communications program and had used the FM routines to read and store the file. I located the listing we used, analyzed the code, and here is the result.

The entire program could have been written in assembler, but since most of my programs are in Applesoft (with machine language support routines), I decided to write it as simple as possible. The name of the file to be transferred, the OPEN, READ, WRITE, and CLOSE commands are all obtained through a short Applesoft front end program. The machine language portion is broken down as follows.

Lines 1130-1150 are simply easily accessible jump vectors to the two routines which will be CALLed from inside an Applesoft driver.

Lines 1190-1320 clear the buffer, in this case $2000-95FF, to zeroes. This gives us a buffer of 30,208 bytes, which should be large enough for most text files. (This is 118 sectors.) Lines 1330-1340 reset the base buffer address, for use later to find the end of the data in the buffer.

Lines 1360-1460 load the file that has been OPENed by the Applesoft driver. The process of setting up a FM parameter block is simplified by using a preset data area called RD.BLK, lines 1790-1800. Calling FM.SETUP sets up the Y- and A-registers properly, and then calling FM.ENTRY reads the text file.

Lines 1500-1580 search through the data buffer for the first occurrence of a 00 byte, which will signal the end of data. By subtracting the base buffer address in lines 1660-1710 we get the actual length of the data. Lines 1600-1650 copy in the initial parameter values for writing, and lines 1660-1710 set up the length.

Lines 1720-1740 call on FM to actually write the data on the file that has been opened in the Applesoft driver.

The time saving using this transfer is significant. A text file containing 8000 bytes took 49 seconds to read and write using pure Applesoft. Using the FM the same operation was

accomplished in only 17 seconds.

Since the program is only 120 bytes long, it can be placed almost anywhere there is free space, especially on page 3. If you are working from a larger Applesoft program, the starting point for the buffer could be moved as needed to load your text file.

```
                    1000 *SAVE POTTS TEXT COPIER
                    1010 *--------------------------------
                    1020        .OR $300
                    1025        .TF TEXT.TRANSFER.OBJ
                    1030 *--------------------------------
2000-               1040 MY.BUFFER  .EQ $2000
                    1050 *--------------------------------
E0-                 1060 BUFFER     .EQ $E0,E1     POINT TO FILE BUFFER
E2-                 1070 RESULT     .EQ $E2        FILE MANAGER RETURN CODE
                    1080 *--------------------------------
03DC-               1090 FM.SETUP   .EQ $3DC       INITIALIZE Y & A
03D6-               1100 FM.ENTRY   .EQ $3D6       FILE MANAGER ENTRY POINT
B5BB-               1110 FM.BLK     .EQ $B5BB      FILE MANAGER PARM LIST
                    1120 *--------------------------------
                    1130 *   SET UP JUMP VECTORS
0300- 4C 06 03      1140        JMP INITIALIZE.AND.READ
0303- 4C 3A 03      1150        JMP FIND.END.AND.WRITE
                    1160 *--------------------------------
                    1170 INITIALIZE.AND.READ
                    1180 *--------------------------------
                    1190 INITIALIZE.THE.BUFFER
0306- A9 00         1200        LDA #MY.BUFFER
0308- 85 E0         1210        STA BUFFER         LSB
030A- A9 20         1220        LDA /MY.BUFFER
030C- 85 E1         1230        STA BUFFER+1       MSB
030E- A0 00         1240        LDY #0
0310- A9 00         1250 .1     LDA #0             CLEAR BUFFER UP TO $95FF
0312- 91 E0         1260 .2     STA (BUFFER),Y
0314- C8            1270        INY                NEXT BYTE IN THIS PAGE
0315- D0 FB         1280        BNE .2             ...STILL IN THE PAGE
0317- E6 E1         1290        INC BUFFER+1       NEXT PAGE
0319- A5 E1         1300        LDA BUFFER+1
031B- C9 96         1310        CMP #$96           AT END OF STORAGE?
031D- D0 F1         1320        BNE .1             ...NO, KEEP CLEARING
031F- A9 20         1330        LDA /MY.BUFFER     RESET BUFFER POINTER
0321- 85 E1         1340        STA BUFFER+1
                    1350 *--------------------------------
                    1360 READ.THE.FILE
0323- A2 09         1370        LDX #9             10 BYTES
0325- BD 6F 03      1380 .1     LDA RD.BLK,X
0328- 9D BB B5      1390        STA FM.BLK,X
032B- CA            1400        DEX
032C- 10 F7         1410        BPL .1
032E- 20 DC 03      1420        JSR FM.SETUP
0331- 20 D6 03      1430        JSR FM.ENTRY
0334- AD C5 B5      1440        LDA FM.BLK+10      GET RETURN CODE
0337- 85 E2         1450        STA RESULT         SAVE FOR APPLESOFT PEEK
0339- 60            1460        RTS                RETURN TO APPLESOFT
                    1470 *--------------------------------
                    1480 FIND.END.AND.WRITE
                    1490 *--------------------------------
                    1500 FIND.END.OF.BUFFER
033A- A0 00         1510        LDY #0             SEARCH FOR 00 BYTE
033C- B1 E0         1520 .1     LDA (BUFFER),Y
033E- F0 07         1530        BEQ .2             ...FOUND END
0340- C8            1540        INY
0341- D0 F9         1550        BNE .1             ...NEXT BYTE IN SAME PAGE
0343- E6 E1         1560        INC BUFFER+1       NEXT PAGE
0345- D0 F5         1570        BNE .1             ...ALWAYS
0347- 84 E0         1580 .2     STY BUFFER         LSB OF EOF BYTE
                    1590 *--------------------------------
                    1600 WRITE.FILE
0349- A2 09         1610        LDX #9             10 BYTES
034B- BD 79 03      1620 .1     LDA WR.BLK,X
034E- 9D BB B5      1630        STA FM.BLK,X
0351- CA            1640        DEX
0352- 10 F7         1650        BPL .1
0354- A5 E0         1660        LDA BUFFER         LSB
0356- 8D C1 B5      1670        STA FM.BLK+6       LSB OF FILE LENGTH
```

```
0359- 38                 SEC
035A- A5 E1              LDA BUFFER+1
035C- E9 20              SBC /MY.BUFFER
035E- 8D C2 B5           STA FM.BLK+7       MSB OF FILE LENGTH
0361- 20 DC 03           JSR FM.SETUP
0364- A2 01              LDX #1             IF NO FILE, ALLOCATE ONE
0366- 20 D6 03           JSR FM.ENTRY       WRITE THE FILE
0369- AD C5 B5           LDA FM.BLK+10      RETURN CODE
036C- 85 E2              STA RESULT         SAVE FOR APPLESOFT PEEK
036E- 60                 RTS                RETURN TO APPLESOFT
                   *--------------------------------
036F- 03 02 00
0372- 00 00 00     RD.BLK .HS 03.02.0000.0000
0375- 00 76 00
0378- 20                 .DA $9600-MY.BUFFER,MY.BUFFER
0379- 04 02 00
037C- 00 00 00     WR.BLK .HS 04.02.0000.0000
037F- 00 76 00
0382- 20                 .DA $9600-MY.BUFFER,MY.BUFFER
                   *--------------------------------
```

```
10  REM  PROGRAM TO TRANSFER A TEXT FILE
20  REM  FROM DRIVE 1 TO DRIVE 2 USING
30  REM  DOS 3.3 FILE MANAGER ROUTINES
40  REM ---BY BOB POTTS. LOUISVILLE, KENTUCKY---
99  REM -------------------
100  HIMEM: 8192: REM  $2000-95FF IS MY BUFFER
110 D$ =  CHR$ (4)
120  PRINT D$"NOMON I,O,C"
130  PRINT D$"BLOAD TEXT.TRANSFER.OBJ"
199  REM -------------------
200 RF = 768: REM CALL ADDRESS TO READ FILE
210 WF = 771: REM CALL ADDRESS TO WRITE FILE
220 RC = 226: REM PEEK ADDRESS FOR FM RETURN CODE
300  REM ----------------------
310  TEXT : HOME : PRINT "TEXT FILE TRANSFER"
320  PRINT "------------------"
330  INPUT "ENTER FILE NAME:  ";F$
400  REM READ FILE FROM DRIVE 1
410  PRINT D$"OPEN"F$",D1
420  PRINT D$"READ"F$
430  CALL RF
440  IF  PEEK (RC) = 5 THEN 500
450  PRINT D$"CLOSE"
460  PRINT "RETURN CODE NOT 'END OF DATA'"
470  STOP
500  REM WRITE FILE TO DRIVE 2
510  PRINT D$"CLOSE"
520  PRINT D$"OPEN"F$",D2
530  PRINT D$"WRITE"F$
540  CALL WF
550  IF  PEEK (RC) = 0 THEN 600
560  PRINT D$"CLOSE"
570  PRINT "RETURN CODE WAS " PEEK (RC)
580  STOP
600  REM FINISHED
610  PRINT D$"CLOSE"
620  PRINT "TRANSFER COMPLETE"
630  END
```

Fast 6502 & 65802 Multiply Routines........Bob Sander-Cederlof

Since multiplication is not a built-in function in the 6502, 65C02, or 65802, many of us have written our own subroutines for the purpose. I will present some efficient subroutines here, to handle the 8-bit and 16-bit cases.

I will assume both arguments are the same length (either 8-bits or 16-bits) and that we want the full product. If the arguments are only 8-bits long, the product will by 16-bits long. If the arguments are 16-bits long, the product will be 32-bits long. I will also assume the arguments are unsigned values. Thus $FF times $FF will be $FE01 (in decimal, 255x255 = 65025).

Way back in February, 1981, I published an article with a Brooke Boering's fast 16-bit multiplication subroutine. His subroutine duplicated the functions of the subroutine in the original Apple Monitor ROM, but was nearly twice as fast. Brooke's programs were originally published in the December, 1980, Micro magazine (now defunct). He included an 8-bit multiply subroutine with an average time of only 192 cycles.

Damon Slye wrote an article for Call APPLE, published June, 1983. He introduced some coding tricks which allow an 8-bit multiply in an average of 160 cycles. I have reproduced Damon's program below, in lines 1010-1300. His trick involves eliminating a CLC opcode from the loop in lines 1210-1260. Ordinarily you would need a CLC before the ADC instruction; Damon decremented the multiplicand by one before starting the loop, so that adding with carry set works. He does the decrementing in lines 1130-1160. Note that if the original multiplicand was zero, he skips all the rest of the code and just returns the answer: 0.

I had to go at least one step faster, so I partially "un-wrapped" the 8-step loop. I changed it to loop only four times, but handled two bits of the multiplier each time. This runs an average time of 140 cycles. You could unwrap it all the way, writing out the BCC-ADC-ROR-ROR lines a total of 8 times, and cut the average time down to only 111 cycles.

Let me stop here and say what I mean by average time. I am stating time in terms of "cycles", rather than seconds or microseconds. The Apple two different cycle times, depending on the video timing logic. The average Apple speed is 1020488 cycles per second. The multiply algorithms will vary in speed depending on the number of bits in the multiplier which equal "1". If the multiplier = $FF (all ones) the algorithm will take the maximum time. If the multiplier is $00, it will take the minimum time. On the average for random arguments, the multiplier will have four zeroes and four ones, so the average time is equal to the average of the minimum and maximum times. For all of the subroutines, I included the cycles for a JSR to call them, and for the RTS at the end.

I programmed an 8-bit multiply using 65802 opcodes, as shown below in lines 1560-1790. The program is slightly shorter (one

byte), but that really isn't a fair comparison. The arguments and product are handled differently, and so the effort to call the program may be more or less than that for the 6502 version. Rather than passing the multiplicand in the X-register, I have it in the A-register. I pass the multiplier in the high byte of the A-register. Since X is not used for passing any values, I saved and restored it (lines 1620 and 1770). I assumed the program would be called from the 6502 mode, which of course it was as long as I was testing it. In "real life" it might be written to be called from Native 65802 mode, since the larger program it was a part of would also be taking advantage of all the 65802 features.

I used a couple of tricks to save space and time. One you may justly complain about is that I store the multiplicand directly into the operand field of the ADC instruction at line 1720. This definitely saves time, but it also could have serious drawbacks. (For example, it would not work if executed from ROM.) Since I enter in 6502 Emulation mode, line 1640 only loads 0 into the low byte of the A-register. Lines 1650-1660 enter the 65802 Native mode. Line 1680 sets the A-register to 16-bit mode.

In line 1690 I form the inverse (one's complement) of the multiplier. This is just another way of eliminating the CLC from the loop. Note that the multiplier is in the high byte of A, and the product is going to be accumulated in the low byte. The loop runs from line 1700 through line 1740. Line 1700 shifts to the left both the partial product and what remains of the multiplier, putting the highest remaining bit of the multiplier into the carry status bit. If that bit = 1, then the original bit in the multiplier before complementing was a zero, so we do not add the multiplicand to the current partial product. As we continue through the loop, the bits of the multiplier keep shifting out just ahead of the ever-growing partial product, until finally we have the answer.

Lines 1750-1780 restore the machine state to the 6502 Emulation mode and restore the original X-register value. The full product is now in the A-register. If I wanted to print out the product, I might do it like this:

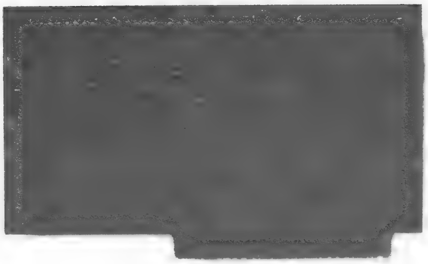
```
       XBA               GET HIGH BYTE INTO LOW-A
       JSR $FDDA         MONITOR PRINT-BYTE SUBROUTINE
       XBA               GET LOW BYTE INTO LOW-A
       JMP $FDDA
```

Here is a summary of the execution times (in cycles) for the three 8-bit multiply subroutines:

	Minimum	Maximum	Average
Slye	152	168	160
RBSC	132	148	140
65802	119	135	127

The 65802 version would be seven cycles faster if we did not require saving and restoring the X-register. If you want to change the 65802 version for calling from Native mode, delete

With Z-80 Plus,™ run CP/M®–the largest body of software in existence.



Now, get two computers in one, and all the advantages of both.

Enter the CP/M world with the new Z-80 Plus card from Applied Engineering, and introduce your Apple IIe® or II+® to the thousands of CP/M programs. Only the Z-80 Plus comes standard with the new 4.0 software, the most advanced system ever for running CP/M programs.

The new 4.0 boasts advanced features like built-in disk emulation for popular memory expansion boards, boosting both system speed and storage capacity. And menu-driven utilities that let you get to work faster. The Z-80 Plus also lets you run older CP/M programs — all the way down to Version 1.6 (2.2 is the most popular).

The Z-80 Plus is the only card on the market capable of accessing more than 64K in an Apple IIe. If you have an extended 80-column card, all 128K is usable, and if you have RamWorks, up to 1088K is available.

Each Z-80 Plus comes with our CP/M Ram Drive software, enabling IIe owners to use an extended 80-column card or a RamWorks card as a high-speed Ram disk which runs CP/M software up to *twenty times faster*. So packages like WordStar and dBASE II run at blinding speed.

Simply plug the Z-80 Plus into any slot in your Apple. You'll get the benefits of two computers in one — all at an unbelievably low price (only $139!).

• Fully compatible with ALL CP/M software • Fully compatible with most hard disks, including Corvus and the Sider • Fully compatible with Microsoft disks (no pre-boot required) • Specifically designed for high speed operation in the Apple IIe (runs just as fast in the Apple II+ and Franklin) • Runs WordStar, dBASE II, Turbo Pascal, Fortran-80, Peachtree and ALL other CP/M software with *no pre-boot* • Semi-custom I.C. and low parts count allows Z-80 Plus to fly through CP/M programs with extremely low power consumption (we use the Z-80B) • Does EVERYTHING other Z-80 boards do, *plus* Z-80 interrupts • Five year warranty

Call to order today, 9 a.m. to 11 p.m. seven days, or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add $10.00 outside U.S.A.

Applied Engineering
P.O. Box 798, Carrollton, TX 75006
(214) 241-6060

Timemaster H.O.,™ the only clock that displays time and date on AppleWorks'™ screens and files.

Now, get all the features of all the competition combined!

It's the smart way to put the time and date on your Apple II+® or IIe®. Because only the Timemaster H.O. packs *ALL* the features of all the competition *combined*, including leap year, year (not just in PRO-DOS), month, date, day of week, hours, minutes, seconds and milliseconds. It's totally PRO-DOS, DOS 3.3, PASCAL and CP/M compatible. And of course, it works better than any other clock with AppleWorks.

If you're using or writing software for other clock cards, you're still covered. Because the H.O. will *automatically* emulate them. And the Timemaster H.O. adds 14 new commands to BASIC. The H.O. even comes complete with two disks full of sample programs, including a computerized appointment book, a DOS dating program, interrupt programs, and over 30 programs that others charge extra for — or don't even offer.

As a low-cost option, you can add true BSR remote control to the H.O., giving you remote control of up to 16 lights and appliances in your home or office.

• Fully PRO-DOS and DOS 3.3, CP/M and PASCAL compatible • Time in hours, minutes, seconds and milliseconds (the ONLY PRO-DOS compatible card with millisecond capability); date with year, month, day of week and leap year • 24-Hour military format or 12-hour AM/PM format • Eight software controlled interrupts so you can run two programs at the same time (many examples included) • Allows AppleWorks to time and date stamp all data automatically • The only clock card that displays time and date on the AppleWorks screen • Five year warranty

Clock price $129.00
BSR option (may be added later) .... $ 49.00

Call to order today, 9 a.m. to 11 p.m. seven days, or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add $10.00 outside U.S.A.

Applied Engineering
P.O. Box 798, Carrollton, TX 75006
(214) 241-6060

lines 1650, 1660, 1750, and 1760. Then insert the following:

```
       1612          PHP
       1614          SEP #$30
       ...
       1772          PLP
```

These changes add one cycle to the time.

```
                        1000 *SAVE S.MULTIPLY 8X8
                        1010 *-------------------------------
02-                     1020 CAND   .EQ 2
03-                     1030 PLIER  .EQ 3
04-                     1040 PROD   .EQ 4,5
                        1050 *-------------------------------
                        1060 *   FAST 6502 MULTIPLICATION, BY DAMON SLYE
                        1070 *   CALL APPLE, JUNE 1983, P45-48.
                        1080 *      (A-REG) = MULTIPLIER
                        1090 *      (X-REG) = MULTIPLICAND
                        1100 *      RETURNS PRODUCT IN A,X (X=LO-BYTE)
                        1110 *-------------------------------
                        1120 FAST.8X8.SLYE
0800- E0 00             1130        CPX #0
0802- F0 17             1140        BEQ .3         A*0=0
0804- CA                1150        DEX            DECR. CAND TO AVOID
0805- 86 02             1160        STX CAND          THE CLC BEFORE ADC CAND
0807- 4A                1170        LSR            PREPARE FIRST BIT
0808- 85 03             1180        STA PLIER
080A- A9 00             1190        LDA #0
080C- A2 08             1200        LDX #8
080E- 90 02             1210 .1     BCC .2         NO ADD
0810- 65 02             1220        ADC CAND
0812- 6A                1230 .2     ROR
0813- 66 03             1240        ROR PLIER
0815- CA                1250        DEX
0816- D0 F6             1260        BNE .1
0818- A6 03             1270        LDX PLIER
081A- 60                1280        RTS
081B- 8A                1290 .3     TXA
081C- 60                1300        RTS
                        1310 *-------------------------------
                        1320 FAST.8X8.RBSC
081D- E0 00             1330        CPX #0
081F- F0 1E             1340        BEQ .3         A*0=0
0821- CA                1350        DEX            DECR. CAND TO AVOID
0822- 86 02             1360        STX CAND          THE CLC BEFORE ADC CAND
0824- 4A                1370        LSR            PREPARE FIRST BIT
0825- 85 03             1380        STA PLIER
0827- A9 00             1390        LDA #0
0829- A2 04             1400        LDX #4
082B- 90 02             1410 .1     BCC .2         NO ADD
082D- 65 02             1420        ADC CAND
082F- 6A                1430 .2     ROR
0830- 66 03             1440        ROR PLIER
0832- 90 02             1450        BCC .25        NO ADD
0834- 65 02             1460        ADC CAND
0836- 6A                1470 .25    ROR
0837- 66 03             1480        ROR PLIER
0839- CA                1490        DEX
083A- D0 EF             1500        BNE .1
083C- A6 03             1510        LDX PLIER
083E- 60                1520        RTS
083F- 8A                1530 .3     TXA
0840- 60                1540        RTS
                        1550 *-------------------------------
                               1560        .OP 65816
                               1570 *-------------------------------
                               1580 *      MULTIPLIER IN A(15-8), MULTIPLICAND IN A(7-0)
                               1590 *      RETURN PRODUCT IN A(15-0)
                               1600 *-------------------------------
                               1610 MULTIPLY.8X8.65802
000841- DA                     1620        PHX
000842- 8D 54 08               1630        STA .2+1       SAVE MULTIPLICAND
000845- A9 00                  1640        LDA #0
```

```
000847- 18          1650        CLC
000848- FB          1660        XCE
000849- A2 08       1670        LDX #8
00084B- C2 20       1680        REP #$20      A-REG 16 BITS
00084D- 49 00 FF    1690        EOR ##$FF00   COMPLEMENT MULTIPLIER
000850- 0A          1700 .1     ASL
000851- B0 03       1710        BCS .3        ...IF ORIGINAL BIT=0
000853- 69 00 00    1720 .2     ADC ##0       ADD MULTIPLICAND
000856- CA          1730 .3     DEX
000857- D0 F7       1740        BNE .1
000859- 38          1750        SEC
00085A- FB          1760        XCE
00085B- FA          1770        PLX
00085C- 60          1780        RTS
                    1790 *--------------------------------
```

I will also show three sample 16-bit multiply subroutines....no, four. The first one is a copy of Brooke Boering's code. The second is a direct conversion of Brooke's code to 65802 code, with emphasis on space. The third modifies the second with the tricks of Damon Slye; it takes more space, but it is faster.

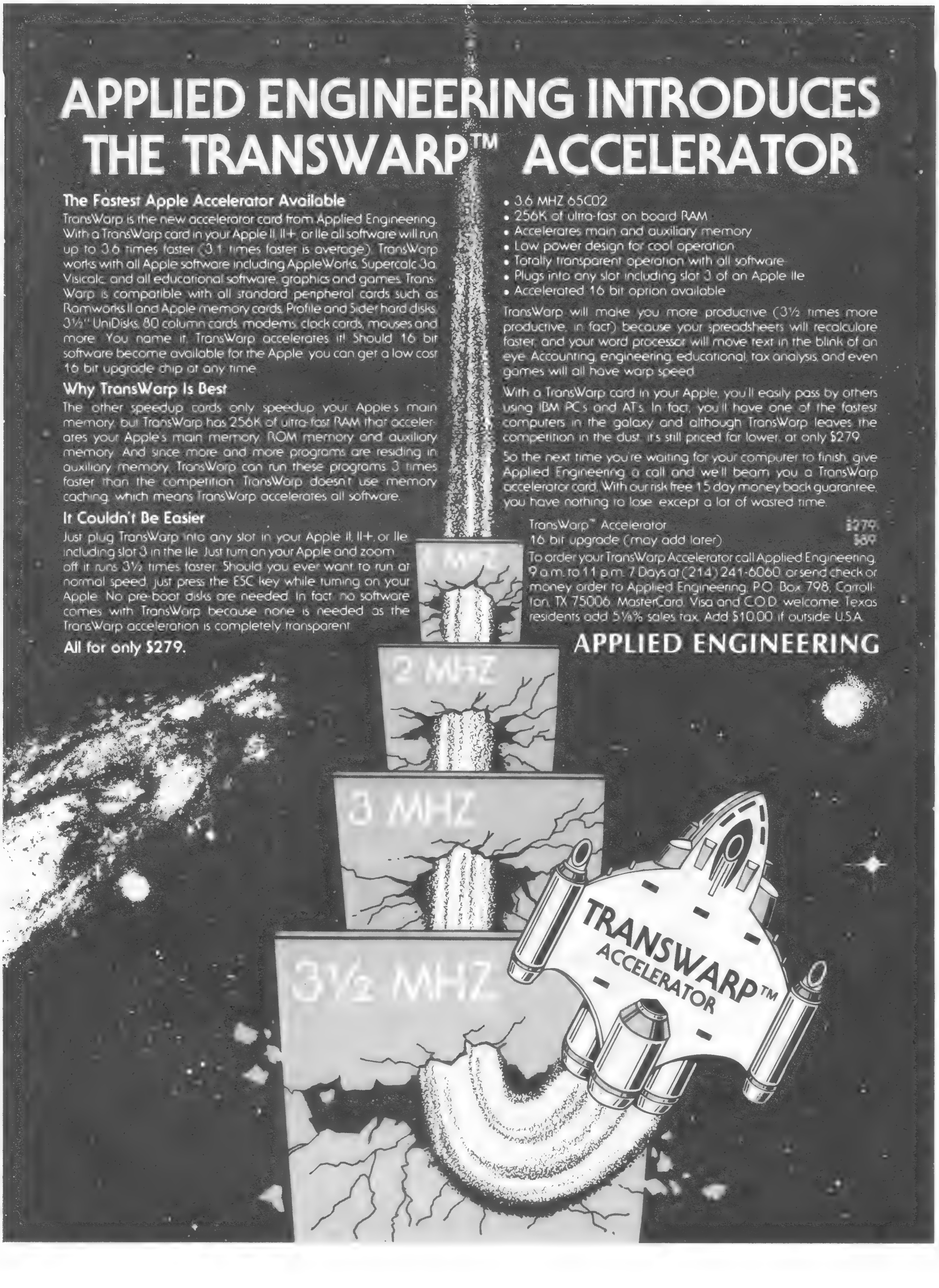
The first three of these subroutines are modeled after the code in the original Apple monitor ROM. The arguments are expected in page zero locations, low-byte first. The result will also be in page zero locations. The function performed is actally a little more than just multiplication, because it is possible to specify an addend as well. The final result will be PRODUCT = ADDEND + (MULTIPLIER * MULTIPLICAND). PRODUCT is stored in four consecutive bytes, backwards. The highest byte is at PRODUCT+1, the next at PRODUCT, the next at PLIER+1, and the lowest at PLIER. The fourth subroutine differs in that the product does not overlap the multiplier.

Looking at Brooke's version (lines 1000-1270) you can see that the loop contains a 16-bit addition (lines 1130-1190). There are also two 16-bit ROR shifts, at lines 1200-1230. These are the likely candidates for shortening via 65802 code. My first version for the 65802 made no other changes in the loop. I merely prefixed Brooke's code with CLC-XCE-REP to get into the 16-bit Native mode, and suffixed it with SEC-XCE to get back to Emulation mode. Then I noticed another shortcut, and the result is in lines 1300-1480.

By moving the LDA PRODUCT up before the BCC opcode in lines 1370-1380, I was able to change a ROR PRODUCT to a simple ROR on the A-register followed by a STA PRODUCT. This saves a net six cycles when the multiplier bit is "1", and costs two cycles when the multiplier bit is "0". The average savings for random multipliers is four cycles, inside a loop that runs 16 times.

The faster version, in lines 1500-1780, merely implements Damon Slye's trick of pre-decrementing the multiplicand so as to avoid an explicit CLC opcode inside the 16-time loop. It costs 12 cycles for the extra setup, but it saves two cycles for each one-bit in the multiplier.

The fourth version, in the separate listing as lines 1000-1430, uses the trick of splitting the multiplier in half. In effect,



APPLIED ENGINEERING INTRODUCES
THE TRANSWARP™ ACCELERATOR
The Fastest Apple Accelerator Available
TransWarp is the new accelerator card from Applied Engineering. With a TransWarp card in your Apple II, II+, or IIe all software will run up to 3.6 times faster (3.1 times faster is average). TransWarp works with all Apple software including AppleWorks, Supercalc 3a, Visicalc, and all educational software, graphics and games. TransWarp is compatible with all standard peripheral cards such as Ramworks II and Apple memory cards, Profile and Sider hard disks, 3½" UniDisks, 80 column cards, modems, clock cards, mouses and more. You name it, TransWarp accelerates it! Should 16 bit software become available for the Apple, you can get a low cost 16 bit upgrade chip at any time.
Why TransWarp Is Best
The other speedup cards only speedup your Apple's main memory, but TransWarp has 256K of ultra-fast RAM that accelerates your Apple's main memory, ROM memory and auxiliary memory. And since more and more programs are residing in auxiliary memory, TransWarp can run these programs 3 times faster than the competition. TransWarp doesn't use memory caching, which means TransWarp accelerates all software.
It Couldn't Be Easier
Just plug TransWarp into any slot in your Apple II, II+, or IIe including slot 3 in the IIe. Just turn on your Apple and zoom off it runs 3½ times faster. Should you ever want to run at normal speed, just press the ESC key while turning on your Apple. No pre-boot disks are needed. In fact, no software comes with TransWarp because none is needed as the TransWarp acceleration is completely transparent.
All for only $279.
• 3.6 MHZ 65C02
• 256K of ultra-fast on board RAM
• Accelerates main and auxiliary memory
• Low power design for cool operation
• Totally transparent operation with all software
• Plugs into any slot including slot 3 of an Apple IIe
• Accelerated 16 bit option available
TransWarp will make you more productive (3½ times more productive, in fact) because your spreadsheets will recalculate faster, and your word processor will move text in the blink of an eye. Accounting, engineering, educational, tax analysis, and even games will all have warp speed.
With a TransWarp card in your Apple, you'll easily pass by others using IBM PC's and AT's. In fact, you'll have one of the fastest computers in the galaxy and although TransWarp leaves the competition in the dust, it's still priced far lower, at only $279.
So the next time you're waiting for your computer to finish, give Applied Engineering a call and we'll beam you a TransWarp accelerator card. With our risk free 15 day money back guarantee, you have nothing to lose, except a lot of wasted time.
TransWarp™ Accelerator $279
16 bit upgrade (may add later) $89
To order your TransWarp Accelerator call Applied Engineering, 9 a.m. to 11 p.m. 7 Days at (214) 241-6060, or send check or money order to Applied Engineering, P.O. Box 798, Carrollton, TX 75006. MasterCard, Visa and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add $10.00 if outside U.S.A.
APPLIED ENGINEERING
2 MHZ
3 MHZ
3½ MHZ
TRANSWARP™
ACCELERATOR

two parallel 8-bit by 16-bit multiplies are accomplished, with the result usually taking less time than any of the other algorithms. By deleting line 1130 (which shaves off another four cycles) the feature of allowing an addend can be included.

Here is a summary of the execution cycles for the four 16-bit multiply subroutines:

	Minimum	Maximum	Average
Boering	541	845	693
Smaller	519	599	559
Faster	531	579	555
Fourth	332	684	508 (usually fastest)

Note that the third subroutine also goes even faster when the multiplicand = zero, because the bulk of the code is skipped.

These are pretty good subroutines, but I have no doubt that they can be improved upon. Why not try your hand? If you can significantly improve either space or time or features, send your code to AAL. We'll publish the best ones, and help advance the state of the art. And if you have some classy division subroutines, they are welcome too!

```
                  1000 *SAVE S.MULTIPLY 16X16
                  1010 *--------------------------------
00-               1020 PLICAND    .EQ $00,01    MULTIPLICAND
02-               1030 PLIER      .EQ $02,03    MULTIPLIER, LO-16 OF PRODUCT
04-               1040 PRODUCT    .EQ $04,05    HI-16 OF PRODUCT
                  1050 *--------------------------------
                  1060        .OP 6502
                  1070 *--------------------------------
                  1080 MULTIPLY.16X16.6502
0800- A2 10       1090        LDX #16
0802- A5 02       1100 .1     LDA PLIER     CHECK NEXT BIT OF MULTIPLIER
0804- 4A          1110        LSR
0805- 90 0D       1120        BCC .2        ...DON'T ADD MULTIPLICAND
0807- 18          1130        CLC
0808- A5 04       1140        LDA PRODUCT
080A- 65 00       1150        ADC PLICAND
080C- 85 04       1160        STA PRODUCT
080E- A5 05       1170        LDA PRODUCT+1
0810- 65 01       1180        ADC PLICAND+1
0812- 85 05       1190        STA PRODUCT+1
0814- 66 05       1200 .2     ROR PRODUCT+1
0816- 66 04       1210        ROR PRODUCT
0818- 66 03       1220        ROR PLIER+1
081A- 66 02       1230        ROR PLIER
081C- CA          1240        DEX
081D- D0 E3       1250        BNE .1
081F- 60          1260        RTS
                  1270 *--------------------------------
                       1280        .OP 65802
                       1290 *--------------------------------
                       1300 MULTIPLY.16X16.65802.SMALLER
000820- 18             1310        CLC
000821- FB             1320        XCE           NATIVE MODE
000822- C2 20          1330        REP #$20      A-REG 16-BITS
000824- A2 10          1340        LDX #16       LOOP 16 TIMES
000826- A5 02          1350 .1     LDA PLIER     CHECK NEXT BIT OF MULTIPLIER
000828- 4A             1360        LSR
000829- A5 04          1370        LDA PRODUCT   GET HI-16 OF PRODUCT
00082B- 90 03          1380        BCC .2        ...DO NOT NEED TO ADD
00082D- 18             1390        CLC
00082E- 65 00          1400        ADC PLICAND
000830- 6A             1410 .2     ROR
000831- 85 04          1420        STA PRODUCT
000833- 66 02          1430        ROR PLIER     USE FOR LO-16 OF PRODUCT
000835- CA             1440        DEX
000836- D0 EE          1450        BNE .1
```

```
000838- 38                     SEC
000839- FB                     XCE          BACK TO EMULATION MODE
00083A- 60                     RTS
                         *-------------------------------
                         MULTIPLY.16X16.65802.FASTER
00083B- 18                     CLC
00083C- FB                     XCE          NATIVE MODE
00083D- C2 20                  REP #$20     A-REG 16-BITS
00083F- A5 00                  LDA PLICAND
000841- F0 19                  BEQ .3       0*ANYTHING=0
000843- 3A                     DEC
000844- 85 00                  STA PLICAND
000846- A2 10                  LDX #16      LOOP 16 TIMES
000848- A5 02            .1    LDA PLIER    CHECK NEXT BIT OF MULTIPLIER
00084A- 4A                     LSR
00084B- A5 04                  LDA PRODUCT  GET HI-16 OF PRODUCT
00084D- 90 02                  BCC .2       ...DO NOT NEED TO ADD
00084F- 65 00                  ADC PLICAND
000851- 6A               .2    ROR
000852- 85 04                  STA PRODUCT
000854- 66 02                  ROR PLIER    USE FOR LO-16 OF PRODUCT
000856- CA                     DEX
000857- D0 EF                  BNE .1
000859- 38                     SEC
00085A- FB                     XCE          BACK TO EMULATION MODE
00085B- 60                     RTS
00085C- A5 04            .3    LDA PRODUCT  INITIAL ADDEND
00085E- 85 02                  STA PLIER    LOW 16 OF PRODUCT
000860- 64 04                  STZ PRODUCT  HIGH 16 OF PRODUCT
000862- 38                     SEC
000863- FB                     XCE          BACK TO EMULATION MODE
000864- 60                     RTS
                         *-------------------------------
```

```
                     *SAVE S.MUL.16X16.65802.EVEN.FASTER
                     *-------------------------------
                               .OP 65802
                         *-------------------------------
00-                      A     .EQ 0.1
02-                      B     .EQ 2.3
04-                      P     .EQ 4,5,6,7
                         *-------------------------------
                         MUL.EVEN.FASTER
000800- 18                     CLC
000801- FB                     XCE          ENTER NATIVE MODE
000802- C2 20                  REP #$20     16-BIT A-REGISTER
000804- 64 06                  STZ P+2      MAKE SURE NO ADDEND IN HI-16
000806- 64 04                  STZ P  (DELETE IF WANT AN ADDEND IN LO-16)
000808- A2 08                  LDX #8
00080A- 80 04                  BRA .2       ...HOP OVER SHIFTS
                         *-------------------------------
00080C- 06 04            .1    ASL P        DOUBLE THE PRODUCT
00080E- 26 06                  ROL P+2
000810- A5 00            .2    LDA A
000812- 29 80 00               AND ##$0080  LOOK AT SIGN OF LO-BYTE
000815- F0 0B                  BEQ .3       ...DON'T ADD MULTIPLICAND
000817- 18                     CLC
000818- A5 04                  LDA P
00081A- 65 02                  ADC B
00081C- 85 04                  STA P
00081E- 90 02                  BCC .3
000820- E6 06                  INC P+2      ADD CARRY TO HI-16
                         *-------------------------------
000822- 06 00            .3    ASL A        SHIFT MULTIPLIER
000824- 90 0B                  BCC .4
000826- 18                     CLC
000827- A5 05                  LDA P+1      ADD TO MIDDLE OF PRODUCT
000829- 65 02                  ADC B
00082B- 85 05                  STA P+1
00082D- 90 02                  BCC .4
00082F- E6 07                  INC P+3      (NEVER BOTHERS P+4)
                         *-------------------------------
000831- CA               .4    DEX
000832- D0 D8                  BNE .1
000834- 38                     SEC
000835- FB                     XCE
000836- 60                     RTS
                         *-------------------------------
```

RAMWORKS Compatible Auxiliary MOVE Routine.........Harvey Brown

Spirit River, Alberta, CANADA

The MOVE routine inside the Apple //c and //e ROM transfers data conveniently to and from the auxiliary 48K area, but it does not work with the upper 16K area. Also, it does not work with the extra banks of RAM available in cards such as the RAMWORKS from Applied Engineering.

I needed that ability, so I wrote my own MOVE subroutine. Mine uses the page at $200 in main RAM as a buffer, to simplify the movement code. If you want to move from an arbitrary bank to another arbitrary bank, my program will require you to use $200 in main RAM as an intermediate buffer. (Somewhat like stopping at Chicago on your way from Toronto to Dallas.) My program also assumes you are always moving exactly 256 bytes (one page). This simplifies the code and the calling sequence, and is probably a reasonable restriction.

The program begins by copying itself into every bank you are using. The bank numbers must be assembled in to the list in lines 1800-1870. Notice that I use bank number $FF to signal the main RAM, and banks from $00 up to signal the banks of Auxiliary RAM. This code needs to reside in the same location in every bank that will be switched on, because when you move from an auxiliary bank to the main RAM that auxiliary bank will be set up so that all RAM reads come from it. This includes reads for the program, so the program had better be there.

Once the program has been initialized, you can JSR MOVE (or JSR $C03 if you want to use a "frozen" entry point) to move a page. At the time of the JSR MOVE, you should have the high byte of the Auxiliary RAM address in the A-register, and the bank number in the X-register. Set carry (SEC opcode) to indicate moving from main $200, or clear carry (CLC opcode) to indicate moving into main $200. Thinking in terms of a ramdisk application, SEC for a write or CLC for a read.

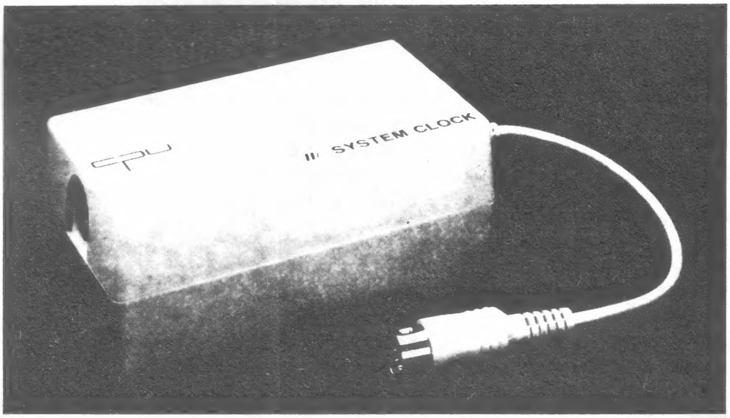
Warning: my program assumes you are calling from a program that runs with the Applesoft ROM selected (see line 1780). If you plan to run it with RAM selected in the upper 16K, you will have to make appropriate changes. You could save the status of the LCRAM and LCBANK soft switches ($C012 and $C011 respectively) before changing them. These partially indicate the status of the $C08x switches. You can tell whether RAM or ROM was selected, and restore the proper one after MOVE is finished. You can also tell which $D000 bank was selected. However, you cannot tell whether the RAM was write-enabled or not; also, you cannot tell if it was in the special mode in which you read ROM and write RAM.

```
                  1000 *SAVE BROWN'S MOVE PROGRAM
                  1010 *-------------------------------
                  1020 * MOVE by H. Brown
                  1030 * Jan 18/86
                  1040 *-------------------------------
00-               1050 PTR    .EQ $00,01
0200-             1060 BUFFER .EQ $200
C002-             1070 RAMRD  .EQ $C002
C004-             1080 RAMWRT .EQ $C004
C008-             1090 ALTZP  .EQ $C008
C073-             1100 BNKSEL .EQ $C073    RAMWORKS BANK SELECT REGISTER
```

```
C082-                  ROM      .EQ $C082
C08B-                  RAM1     .EQ $C08B
C083-                  RAM2     .EQ $C083
                       *--------------------------------
                                .OR $C00      ORG  AT BEGINNING OF A PAGE
                       *--------------------------------
                       COMMONPG
0C00- 4C 06 0C                  JMP INIT      BRUN OR JSR TO INITIALIZE
0C03- 4C 22 0C                  JMP MOVE      NORMAL ENTRY
                       *--------------------------------
                       *   INIT copies COMMONPG to all 64K banks
                       *--------------------------------
0C06- AE 78 0C         INIT     LDX BANKS     START WITH LAST 64K BANK
0C09- BD 78 0C         .1       LDA BANKS,X   GET  BANK #
0C0C- 8D 73 C0                  STA BNKSEL    SELECT 64K BANK
0C0F- 8D 05 C0                  STA RAMWRT+1  CHOOSE TO WRITE
0C12- A0 00                     LDY #0        COPY PAGE
0C14- B9 00 0C         .2       LDA COMMONPG,Y
0C17- 99 00 0C                  STA COMMONPG,Y
0C1A- C8                        INY
0C1B- D0 F7                     BNE .2        LOOP TO END OF PAGE
0C1D- CA                        DEX
0C1E- D0 E9                     BNE .1        LOOP TO START OF TABLE
0C20- F0 46                     BEQ EXIT      RESTORE STANDARD MEMORY
                       *--------------------------------
                       *   enter MOVE with A = page (CX for 2nd DX)
                       *                   X = 64K bank #
                       *                   Carry SET for write, CLEAR for read
                       *--------------------------------
0C22- B0 23            MOVE     BCS .3        BRANCH IF WRITING
0C24- C9 C0                     CMP #$C0
0C26- B0 0C                     BCS .1        BRANCH IF UPPER 16K
0C28- E0 FF                     CPX #$FF      ---  READ 48K ---
0C2A- F0 15                     BEQ .2        SKIP IF MAIN 64K
0C2C- 8D 03 C0                  STA RAMRD+1   READ FROM AUX 48K
0C2F- 8E 73 C0                  STX BNKSEL    SELECT 64K BANK
0C32- D0 0D                     BNE .2
0C34- 20 98 0C         .1       JSR SEL16K    ---  READ 16K ---
0C37- E0 FF                     CPX #$FF
0C39- F0 06                     BEQ .2        SKIP IF MAIN 64K
0C3B- 8E 73 C0                  STX BNKSEL    SELECT 64K BANK
0C3E- 8D 09 C0                  STA ALTZP+1   SELECT AUX 16K
0C41- 18               .2       CLC
0C42- 20 7E 0C                  JSR COPYPAGE
0C45- F0 21                     BEQ EXIT
                       *--------------------------------
                       *   WRITING
                       *--------------------------------
0C47- C9 C0            .3       CMP #$C0
0C49- B0 0C                     BCS .4        BRANCH IF UPPER 16K
0C4B- E0 FF                     CPX #$FF      ---  WRITE 48K ---
0C4D- F0 15                     BEQ .5        SKIP IF MAIN 64K
0C4F- 8E 73 C0                  STX BNKSEL
0C52- 8D 05 C0                  STA RAMWRT+1  WRITING TO AUX 48K
0C55- D0 0D                     BNE .5
0C57- 20 98 0C         .4       JSR SEL16K    ---  WRITE 16K ---
0C5A- E0 FF                     CPX #$FF
0C5C- F0 06                     BEQ .5
0C5E- 8E 73 C0                  STX BNKSEL
0C61- 8D 09 C0                  STA ALTZP+1
0C64- 38               .5       SEC
0C65- 20 7E 0C                  JSR COPYPAGE
                       *--------------------------------
0C68- 8C 73 C0         EXIT     STY BNKSEL    RESORE STD 64K FOR VIDEO
0C6B- 8D 04 C0                  STA RAMWRT    MAIN 48K
0C6E- 8D 02 C0                  STA RAMRD
0C71- 8D 08 C0                  STA ALTZP     MAIN 16K
0C74- AD 82 C0                  LDA ROM
0C77- 60                        RTS
                       *--------------------------------
                       *   BANKS is a table of 64K bank #'s, where
                       *       FF = main 64k, 00 = alt 64K when no RAMWORKS
                       *       00,04,08,0C = banks of a 256K RAMworks
                       *       1st entry is # of banks
                       *--------------------------------
0C78- 05               BANKS    .HS 05        Five banks all told
0C79- FF 00 04
0C7C- 08 0C                     .HS FF.00.04.08.0C
```

```
                1880 *--------------------------------
                1890 *   COPYPAGE copies 256 bytes
                1900 *      from (PTR) in specified bank to motherboard $200
                1910 *   or from motherboard $200 to (PTR) in specified bank
                1920 *--------------------------------
                1930 COPYPAGE
0C7E- 85 01     1940        STA PTR+1
0C80- A0 00     1950        LDY #0
0C82- 84 00     1960        STY PTR
0C84- B0 09     1970        BCS .2
0C86- B1 00     1980 .1     LDA (PTR),Y
0C88- 99 00 02  1990        STA BUFFER,Y
0C8B- C8        2000        INY
0C8C- D0 F8     2010        BNE .1
0C8E- 60        2020        RTS
0C8F- B9 00 02  2030 .2     LDA BUFFER,Y
0C92- 91 00     2040        STA (PTR),Y
0C94- C8        2050        INY
0C95- D0 F8     2060        BNE .2
0C97- 60        2070        RTS
                2080 *--------------------------------
                2090 *   SEL16K selects the appropriate bank in 16K area
                2100 *--------------------------------
0C98- C9 D0     2110 SEL16K CMP #$D0
0C9A- B0 09     2120        BCS .1
0C9C- AC 83 C0  2130        LDY RAM2     C0   -> AUX D0
0C9F- AC 83 C0  2140        LDY RAM2
0CA2- 69 10     2150        ADC #$10
0CA4- 60        2160        RTS
0CA5- AC 8B C0  2170 .1     LDY RAM1     SELECT RD/WRT RAM
0CA8- AC 8B C0  2180        LDY RAM1
0CAB- 60        2190        RTS
                2200 *--------------------------------
```

NEW PRODUCT



IIc SYSTEM CLOCK

- Fully ProDos compatible
- Automatic time and date stamping
- Easy to use from BASIC
- Battery operated, uses 3 "AA" batteries (will last 1-2 years before simple replacement)
- Date has year, month, date and day of week
- Time has hours, minutes and seconds
- Will time and date stamp AppleWorks files
- Will display time and date on the AppleWorks screen
- Auto access from AppleWorks data-base (just use a time and date field)
- Pass through serial port - The IIc system clock can plug into either the modem or printer serial port, then modem or printer plugs into the clock
- No hassle 5 year warranty
- Only $79.00

Æ

"We Set the Standard"

214-241-6060 APPLIED ENGINEERING 9 AM - 11 PM

Correction to DOS/ProDOS Double Init.......Bob Sander-Cederlof

The Sep 85 (V5N12) issue of AAL included an article and program to initialize a disk with both DOS and ProDOS catalogs in separate halves of the disk. After trying to use Catalog Arranger on a disk we made with DOUBLE.INIT, we discovered that program has a bug.

The DOS catalog is written in track $11, starting with sector 15 and going backwards to sector 1. The second and third bytes in each catalog sector are supposed to point to the next catalog sector, with the exception of those bytes in the LAST catalog sector. In the last catalog sector, the link bytes should both be $00, to signal to anyone who tries to read the catalog that this is indeed the last sector. DOUBLE.INIT stored $11 in the first link byte, and so some catalog reading programs such as Catalog Arranger get very confused.

The fix is to add the following lines to the program, where the line numbers correspond to those in the printed listing in AAL:

```
       2201       BNE .5
       2202       STY C.TRACK    (Y=0)
```

Add the label ".5" to line 2210, so that it reads:

```
       2210 .5    JSR CALL.RWTS
```

If you have already created some disks with DOUBLE.INIT, we suggest you use a program such as Bag of Tricks, CIA, or some other disk zap program to clear the second byte of track $11, sector $01 on those disks.

An Interesting Bit of Trivia........................Bill Parker

Some time ago I asked Bob S-C if he knew the origin of the term "6502". Why was this particular number chosen? Bob didn't know, but referred me to Bill Mensch at Western Design Center.

Bill worked at Motorola and was on the design team that created the 6800, which later led to the development of the 68000. He left Motorola with a few others and formed MOS Technology (now absorbed into Commodore), where they developed a new microprocessor which was supposed to be an improved version of the 6800. Hence the decision was made to use a number in the 6000 series. As for the hundreds digit, Commodore already had chips that used just about every digit, except "5". Thus, the "6500" series of chips was born.

As history tells us, the first chip in the series, the 6501, was too close to Motorola's design, and had to be revised. The result was the 6502.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).